Transfer-Learning Methods in Programming Course Outcome Prediction

Lagus, Jarkko

2018-11

Lagus , J , Longi , K , Klami , A & Hellas , A 2018 , ' Transfer-Learning Methods in Programming Course Outcome Prediction ' , ACM Transactions on Computing Education , vol. 18 , no. 4 , pp. 19:1-19:18 . https://doi.org/10.1145/3152714

http://hdl.handle.net/10138/323306
https://doi.org/10.1145/3152714

acceptedVersion

Transfer-Learning Methods in Programming Course Outcome Prediction

JARKKO LAGUS, KRISTA LONGI, ARTO KLAMI, and ARTO HELLAS,
University of Helsinki, Finland

The computing education research literature contains a wide variety of methods that can be used to identify students who are either at risk of failing their studies or who could bene t from additional challenges. Many of these are based on machine-learning models that learn to make predictions based on previously observed data. However, in educational contexts, di erences between courses set huge challenges for the generalizability of these methods. For example, traditional machine-learning methods assume identical distribution in all data in our terms, traditional machine-learning methods assume that all teaching contexts are alike. In practice, data collected from di erent courses can be very di erent as a variety of factors may change, including grading, materials, teaching approach, and the students.

Transfer-learning methodologies have been created to address this challenge. They relax the strict assumption of identical distribution for training and test data. Some similarity between the contexts is still needed for e cient learning. In this work, we review the concept of transfer learning especially for the purpose of predicting the outcome of an introductory programming course and contrast the results with those from traditional machine-learning methods. The methods are evaluated using data collected in situ from two separate introductory programming courses.

We empirically show that transfer-learning methods are able to improve the predictions, especially in cases with limited amount of training data, for example, when making early predictions for a new context. The di erence in predictive power is, however, rather subtle, and traditional machine-learning models can be su ciently accurate assuming the contexts are closely related and the features describing the student activity are carefully chosen to be insensitive to the ne di erences.

CCS Concepts: Social and professional topics Computer science education; Computing methodologies Machine learning; Transfer learning;

Additional Key Words and Phrases: Machine learning, transfer learning, source code snapshots, introductory programming, educational data mining, learning analytics, novice programmers, course outcome prediction

ACM Reference format:
Jarkko Lagus, Krista Longi, Arto Klami, and Arto Hellas. 2018. Transfer-Learning Methods in Programming Course Outcome Prediction. ACM Trans. Comput. Educ. 18, 4, Article 19 (July 2018), 18 pages.
https://doi.org/10.1145/3152714

This work was supported in part by the Academy of Finland (decision numbers 251170 and 266969).
Authors addresses: J. Lagus, K. Longi, A. Klami, and A. Hellas, University of Helsinki, Department of Computer Science, P. O. Box 68, Helsinki, FI-00014, Finland; email: rstname.lastname@helsinki. .

https://doi.org/10.1145/3152714

1 INTRODUCTION

Educational sciences have started to adopt machine-learning methods, computer algorithms that learn predictive models based on observed data. These methods have been applied in research to, for example, automate tutoring [17], predict course outcomes [23], identify at-risk students [1], detect student characteristics [7], predict learning disabilities [25, 40], model students learning process [30], and much more. Many of these tasks can be performed transparently, as they only use data that can be naturally gathered from existing learning environments.

One stream of such research is related to the analysis of source code snapshots, i.e., process data, that is recorded as students work on programming assignments [18]. Typically, the models are applied in post hoc manner for analyzing data from a single course and semester, with only a few studies seeking to perform cross-course or cross-context analysis [18]. Analyzing multiple contexts, such as the same course during different semesters or similar courses in different universities, in combination has two obvious advantages: It increases the amount of available data for learning, and it allows us to study relationships between different contexts.

Cross-context datasets can, however, be problematic from the point of view of traditional machine-learning methods. The core principle of machine learning is that the predictive model is learned from some observations collected for training the model, and, consequently, the algorithms learn to make predictions for future data instances that are similar to those used for training the model—in technical terms, they are assumed to be drawn from the same probability distribution. The more this assumption is violated the less useful the models become. If the contexts are sufficiently different, then it can be better to ignore the different contexts altogether. Indeed, using machine-learning methods to construct a predictive model on data from one semester and using it to predict the outcomes of another semester has been observed to produce worse predictions than using data from the same semester for both learning the model and making the predictions [1, 6].

In this work, which is motivated for the need of context-independent methodologies [18, 29], we study whether transfer-learning methodologies could be used for the task of predicting students' outcomes in a cross-context setting. Transfer learning is an area of machine learning that deals with data coming from multiple sources that may have different distributions in data, e.g., different programming assignments, teaching approaches, and so on, and combines that information to improve the prediction model of the task at hand. Naturally, as is often the case with machine-learning approaches, transfer learning relies on having at least some data with values—i.e., course outcomes—from all contexts.

Our analysis is based on two introductory programming courses. One of the courses had lectures and individual assignments, while the other had no lectures, and the students had both pairwise and individual assignments. The grading scheme of both courses also differs from each other.

This article is organized as follows. First, in Section 2, we provide a brief overview of machine learning and transfer learning and visit the related work on predicting course outcomes. Then, in Section 3 we discuss our research methodology and data in more detail, followed by the results of our study in Section 4. The results are discussed in Section 5, where we also address the main limitations of our work. Finally, in Section 6, we briefly review the main contributions of this article and outline possible future directions for study.

2 MACHINE-LEARNING METHODS AND PREDICTION OF COURSE OUTCOMES

In this section, we first provide a brief overview of machine learning with the focus on supervised machine-learning methods and how they are evaluated. We then proceed to describe in more detail the concept of transfer learning and practical methods for it. Finally, we provide an overview of related studies that have sought to predict students' performance in programming courses using machine-learning methods.

2.1 Overview of Machine Learning

The Merriam-Webster online dictionary [24] de nes learning as the activity or process of gaining knowledge or skill by studying, practicing, being taught, or experiencing something: the activity of someone who learns. Machine learning, in contrast, is based on de ning a learning algorithm that provides guidelines for the machine, but the actual model is learned from a collection of observations (called data). In other words, the machine has access to a set of examples and its task is to learn a generic model that can, to some extent, make predictions that are similar to those examples. Machine-learning methods are typically applied when it is too di cult to compose an exact algorithm for a particular task. This is also where machine learning di ers from traditional algorithm design, where the decision process of the algorithm is encoded manually to the program.

For the purpose of course outcome predictions, the most relevant type of machine learning is called supervised machine learning. The training data are provided as pairs of inputs and outputs, and the task of the learning algorithm is to provide the output for any future input. For example, we can use the knowledge of which assignments a student solved during the rst half of the course as the input and the nal course grade as the output. After training a machine-learning model based on a collection of such pairs, we can use it for predicting the grade for any future student at halfway through the course.

2.1.1 Data and Features.

Machine-learning models are trained on observed data. Furthermore, data quality and quantity strongly in uence what can be achieved if we are only given the age of the student, we cannot hope to make accurate predictions about his or her study performance. However, a rich representation of the students past performance and activity during a course should be su cient for creating somewhat accurate predictions of the nal exam performance.

In typical modeling scenarios the data are provided as a set of training instances, here students. The information available for each student is represented as a feature vector, a set of features that each describe a particular property of that student. The features can be either background information, such as the student s year of birth or whether the student has previous experience on the topic, or they may be extracted from data that a course management system collects as the student works on course tasks here, a typical feature could be whether the student completed a particular assignment or not.

Typically, the features used for training the model are a result of manual engineering e ort that aims to produce a set of features that are maximally rich in capturing the performance of the students, while still being reliable and easy to estimate. To identify relevant features, we can perform algorithmic feature selection to extract a more compact description, discarding features initially thought to be relevant but that do not really help in solving the learning task. Feature selection can be done with respect to the variable that we are predicting, or in a more generic way where one does not care about the target variable but seeks to maximize the overall information in the data [15]. Highly correlated features are also often removed as much of their information is shared in such a case, only one of the features may be needed to catch most of the relevant information in a set of features [16].

2.1.2 Learning and Evaluating the Model.

The process of machine-learning can be summarized as (1) speci cation of the model and algorithm, (2) speci cation of the learning goal, and (3) learning the parameters of the model. For classi cation, the learning goal is typically to minimize the number of errors made by the algorithm. Learning the parameters of the model means that the machine-learning algorithm seeks to identify parameters that minimize the error.

This is an optimization problem, the details of which heavily depend on the machine-learning algorithm in question. A core di erentiation from classical optimization literature, however,

Table 1. A Contingency Table of a Classification Task Where Each Instance Is Categorized Either into Class True or Class False

		Predicted	
		Class = True	Class = False
Actual	Class = True	TP	FN
	Class = False	FP	TN

concerns the nature of the learning goal the nal goal of machine learning is not to minimize errors in training but instead to minimize the errors for future test instances (generalization error). For nite data collections, the lowest generalization error is typically not obtained by the model that minimizes the training error, but instead we need to regularize the model to avoid over tting solutions that describe the training data well but do not generalize to test instances.

To tackle over tting, the data are usually divided into training and validation sets. Training data are used to build the model, and the goodness of the model is evaluated with a separate validation set to recognize models that have low generalization error. One of the most often used methods for model validation is cross-validation, where a xed number of partitions, folds, of the data is chosen. Then, iterating over all partitions, one of the partitions is used for validation, and the rest is used for training. The nal performance estimate is calculated as the average of these iterations. Research suggests the use of 10 as the partition count [39].

2.1.3 Performance Metrics. The evaluation process described above is carried out with respect to a performance metric that quanti es the quality of the solution. One of the most used ways to visualize performance outcomes when evaluating classi cation results is the confusion matrix (see Table 1). It is a table that has each class once on both vertical and horizontal axis. As it lists the full classi cation outcome, it also makes it straightforward to visualize potential imbalances in datasets and to see how the predictions are distributed among di erent classes.

Table 1 uses abbreviations TP for the number of true positives (positive instances classi ed correctly), TN for the number of true negatives (negative instances classi ed correctly), FP for the number of false positives (negative instances classi ed incorrectly as positive ones), and FN for the number of false negatives (positive instances classi ed incorrectly as negative ones).

The confusion matrix can be used to calculate various types of performance metrics. One commonly used metric is the F1-score (Equation (3)), which is a weighted average of precision (Equation (1)) and recall (Equation (2)).

$$\text{Precision} = \frac{TP}{TP + FP}, \tag{1}$$

$$\text{Recall} = \frac{TP}{TP + FN}, \tag{2}$$

$$F1 = \frac{2TP}{2TP + FP + FN}. \tag{3}$$

2.2 Classification Methods

Next, we give a brief overview to the classi cation methods applied in this work. The methods are chosen as a representative sample of the classi ers that have recently been suggested to perform well for the task of classifying students. At the same time, we also seek a balance with the quantity

of classi ers, as our rst and foremost goal is not to provide a full classi er comparison but to evaluate the feasibility of transfer-learning methods for our task.

2.2.1 Support Vector Machines. Support vector machine (SVM) is a classi er that builds a set of margins from the training data that best discriminate between the outcomes [11]. In the case of a linearly separable data, linear classi ers can be used. A linear classi er is a geometric model that uses a linear function to construct the decision border from the margins [11]. However, if the data are not linearly separable, then kernel methods can be used as a part of the support vector machine. Kernel methods transform the data points to a new space, where the instances are linearly separable.

2.2.2 Random Forests, Decision Trees, and Bagging. Decision trees are treelike graphs that represent sequences of classi cation rules. Every node contains a rule that splits the tree into subtrees, and the leaf node de nes the nal class. Random forest is a collection of such decision trees combined using a technique called bagging.

Bagging, short for bootstrap aggregating, creates multiple random samples with replacement from the training data [39]. These random samples are then used to train a set of classi ers that are used in the nal model. When the nal model is used for predicting the outcomes for new data, the actual prediction is based on a vote over all the classi ers in the model.

Random forests use bagging on decision trees to prevent some of the inherent problems with decision trees. In decision trees, small changes in training data may result in very di erent models as a change in top-most rules can alter the structure of the whole subtree [39]. Thus, decision trees are also prone to over- tting. As shown by Breiman [4], random forests avoid the problem of over- tting and improve in accuracy when more trees are added.

2.2.3 Boosting. Boosting, like bagging, is based on the idea of using a set of classi ers for building the model. In Boosting, classi ers complement each other and avoid repeating already-known knowledge. If a classi er performs poorly with speci c values in the data, then another classi er should compensate the lack of performance and vice versa [39].

Our focus here is on a variant of boosting that is also used in transfer-learning algorithms described later. AdaBoost [12] (short for Adaptive Boosting) is one of the most widely used boosting algorithms. It is based on the idea of iteratively assigning weights on training data instances. When learning a new classi er higher weight is given for instances that are misclassi ed by the existing set of classi ers, so that the new classi er best complements the ones trained earlier. AdaBoost performs weighting also on another level: The predictions are averaged over the set of classi ers and more weight is given for classi ers with high accuracy [33].

AdaBoost is not a learning algorithm itself, but a method that uses a collection of some learning algorithms. The only requirement for the learning algorithm AdaBoost is used with is that it should support using weights on training instances.

2.3 Transfer Learning

Next, we discuss what transfer learning is, what separates it from traditional supervised machine learning, and when and why it is used. Then, we introduce two transfer-learning algorithms, TrAdaBoost and TransferBoost.

2.3.1 Overview. Traditional machine-learning methodologies are based on the assumption that the training and testing data come from the same distribution. This can lead to a situation where even small changes in the distribution can render the original model unusable for future uses or, at least, lower its performance signi cantly. Moving away from this assumption is the central goal of transfer learning.

The aim of transfer learning is to extract knowledge from related tasks, called source tasks, to be used for improving the predictive model of the task we are trying to learn, a target task. Transfer learning is applied, for example, in situations where a large collection of data exists for a suitable source task but there is not enough data from the target task available to create an accurate model, or when collecting a new dataset is too expensive and time consuming [26].

For the knowledge transfer to be successful, there has to be some common ground between the source and target task. For example, in the educational context, the courses usually stay the same, but students and lecturer might change between course iterations. The traditional approaches for modeling the outcomes for a new iteration are based on either re-using a model trained on previous iterations, or training the model from scratch on data collected during the early stages of the new iteration. The latter approach su ers from limited data whereas the former may perform badly if the data di ers too much from the previous iterations. Transfer learning addresses this challenge by providing mechanisms that use both the previous (source data) and current (target data) course instances to train the models, and even allows using multiple previous iterations as separate source datasets that need not be assumed identical. Studies show that it is almost always bene cial to use multiple sources of data instead of just a single source [10, 41].

Many of the existing transfer-learning algorithms are based on AdaBoost, which was brie y discussed previously. The main di erence in transfer-learning adaptations is that instead of creating a model that contains multiple classi ers from the same data, multiple datasets are used to build the classi ers.

2.3.2 TrAdaBoost and TransferBoost. In instance-transfer, which is a type of transfer learning, previously acquired data instances are used, and transfer is conducted through reweighing, including or excluding the data instances. The goal is to choose those data instances that are relevant for the target domain, and to give less weight to those instances that are unlikely to be relevant for the target domain [10].

TrAdaBoost [9] and TransferBoost [10] are both based on instance-transfer. TrAdaBoost uses AdaBoost for the target data, and then weights the classi ers built from the source data to reduce the weight of misclassi ed source instances to focus more on the target domain [10]. As a part of reducing the weights, TrAdaBoost discards a proportion of the classi ers to ensure convergence.

Eaton and desJardins argue that discarding a proportion of the classi ers may degrade the empirical performance of the classi er [10]. TransferBoost is a generalization of the AdaBoost algorithm that supports multiple data sources. Similarly to TrAdaBoost, TransferBoost tries to weight the samples that match to the distribution of the target domain. The main di erence to TrAdaBoost is that TransferBoost assigns a importance weight for each data sample. The importance weights are an estimate for how well the source data can be transferred over to the new domain. They call this measure transferability [10]. By introducing this additional importance weight they show that by choosing the dataset and the classi er weights cleverly (for details, see References [10]), the training loss can be bounded to a reasonably low upper-bound even when all the classi ers are included, thus overcoming the problem of having to discard classi ers.

2.4 Predicting Course Outcomes

Next, we brie y visit recent literature that is related to predicting programming course outcomes. In most of the cases, outcome prediction focuses on course grades or on whether the student passes the course, but in some cases, researchers may also focus on the outcomes of individual assignments. For a more extensive review, see, e.g., References [1, 18].

One of the cornerstones of the recent work on using process data for predicting students course outcomes is the error quotient algorithm [19]. Error quotient is based on the idea that students who

are likely to perform poorly struggle to x consecutive syntax errors, while students who are likely to perform well do not struggle as much. It is one of the few algorithms that has been studied in multiple contexts [20, 29].

Error quotient and its relatives (see, e.g., References [2, 5, 38]) are approaches where the data used for the outcome prediction has required signi cant insight into the programming process and the internals of the used programming language. This is not necessary, however, as one can also use higher level information such as the number of actions or attempts that a student makes on assignments as a feature [1, 6]. It is also possible to extend this thinking outside the programming environment and, for example, use data extracted from students in-class behavior such as clicker answers. This was recently proposed by Reference [31] and later extended by Reference [21].

While there exists much work on the topic, so far, there has been little work on methods that study whether speci c methods work in di erent contexts [18]. Some of the exceptions include the work by Petersen et al., who explored the performance of the error quotient algorithm in three di erent contexts [29], and the study in Reference [6], who explored the applicability of neural networks to predicting cross-semester course outcomes.

3 METHODOLOGY

3.1 Research estions

The overall theme in our research is predicting whether students will fail or pass an introductory programming course. For this task, we focus on the applicability of data from previous and somewhat dissimilar course instances for the outcome prediction, and study whether transfer-learning methods would be bene cial applicable for the task. Our research questions are as follows:

(1) How do traditionally used machine-learning methods perform in identifying at-risk students over course instances?
(2) Can this performance be improved with the use of transfer-learning methods? If so, then how large is the impact?

With the rst research question, we both revisit a previous study from Ahadi et al. [1], as well as form a baseline that shows how machine-learning methods fare when predicting course outcomes in somewhat dissimilar contexts. We study two separate course outcome possibilities, where we rst seek to identify those who pass the course and then seek to identify those with overall course points over the course median.

This is then followed by the second research question, where transfer-learning methods are applied to the same data, with the purposes of determining the applicability of transfer-learning methodologies to our problem.

3.2 Context and Data

We study the question of transfer learning in a setup consisting of two courses, mimicking a typical scenario for which transfer learning is expected to be useful. We have collected a large amount of training data from a past course and want to apply the model for a new version of the course for which we can only access a small amount of training examples, obtained, for example, by running a quick pilot with only a few students (or a near-parallel course, see, e.g., Reference [36]). In practice, we conduct the study in a post hoc so that we have large number of students also for the latter course (see the details below) but simulate this scenario by only using a subset of the students in the latter course for training the model.

The data for this study come from two introductory Java programming courses organized at the University of Helsinki. The main di erences in the courses were related to lectures,

Table 2. Summary of the Course Di erences

	Course 1 (source)	Course 2 (target)
Number of students in dataset	247	101
Of whom passed the course	193	66
Pass-rate	78%	65%
Percentage CS majors	30%	5%
Grading	Linear 0 5	pass/fail
Minimum course points to pass	Over 50%	Over 70%
Exam cuto	Yes	Yes
Lectures	Yes	No
Individual assignments	Yes	Yes
Weekly (unguided) lab-hours	45	20
Pair-programming	No	Yes

course activities, and grading, but both courses were 7 weeks in length and covered the typical introductory programming course contents such as variables, input and output, lists, methods, objects and elementary sorting and searching algorithms. The material for the second course had extra assignments and additional explanations, but no signi cant changes were done in the material between the courses.

The rst course had weekly lectures that used active learning methodologies as a part of the lectures. The course had 45 weekly lab hours where students could come and work on course assignments under guidance from course assistants and instructors. The second course had no lectures but had 20 weekly lab hours where students could come to work on the course assignments. Both courses had weekly individual assignments, and the second course also had weekly pair programming assignments.

The rst course was graded on a linear scale from 0 to 5, where 0 would mean failing the course (less than or equal to 50% of overall points), and 5 would equal the best possible grade (over 90% of overall points). The second course was graded on a fail-pass -scale, where students would have to reach a grade comparable to 3 from the rst course to pass the course (over 70% of overall points). In both courses, roughly 1/3 of the course points came from a written exam, and 2/3 of the course points came from completing the course assignments. Finally, an exam cuto was in place in both courses, which meant that the exam had to be completed with at least half of the overall points for the students to pass the course. The exam was similar in both courses.

The data from the rst course have information on 247 students (78% pass rate), while the data from the second course has information on 101 students (65% pass rate). From the rst course, 30% of the participants had CS as their major subject, while in the second course, only 5% of the participants had CS as their major. The di erences between the two courses are summarized in Table 2.

As the students worked on the individual assignments, their working process was recorded using Test My Code [37], which is an IDE plugin that the courses use for automatic assessment. The plugin also helps students download course-speci c programming assignments and records the students working process (key presses, timestamps, assignment details) as the students work on the assignments.

At the beginning of the course, the students completed a survey that asked for their background details. The details include year of birth, gender, information on whether they are currently

working, and previous programming experience. As pair programming was done in labs, where only one student is logged in on a computer at a time, no data from pair programming was available for this analysis.

3.3 Data Preprocessing

The data from the students working process were transformed into a form that contains assignment-specific information on events (actions in the IDE), states in which the source compiles, and whether the student was able to complete the assignment. The use of such data was motivated by recent efforts to use programming process data to predict course outcomes (see, e.g., References [1, 18]); information on the proportion of compiling states was included based on Reference [19]. This programming process data wwere then aggregated to the week level to allow better generalizability, as the quantity of assignments can vary between courses.

The working process data were augmented with the background survey details. The background survey was constructed in house and inspired by the surveys in References [1, 38]. Some details such as whether the student has Computer Science as a major were included due to our specific context in the studied context, students pick their major before entering the University, and as such, choosing Computer Science as a major could indicate motivation. Similarly, information of part-time job and employment was motivated in Reference [3] while their work did not show a significant difference between groups that have a part-time job and do not have a part-time job, our experiences suggested that this could differ in our context. When constructing the survey, we sought to identify a balance between the quantity of included questions and their approximated predictive value.

As the survey included the option of typing in free-form text in places where numbers were expected (largest program written in lines, hours spent on programming, working hours per week), manual cleaning was conducted to transform phrases such as maybe 100 and thousands of hours into numeric versions. Here, vague statements such as the maybe 100 were encoded to match the exact number, while plurals were mapped to double the given number, i.e., thousands of hours was mapped to 2000. Then, the non-numeric background features such as gender and major were transformed into binary features. Finally, missing background values were mapped to zeros as the used algorithm had no support for missing values. Even though the background features are only a small subset of all possible features, this can create bias, as also in the binary case the actual responses do take values of zero and 1.

Two separate datasets were constructed. In the first dataset, the course outcomes were shown as pass/fail, where the values from 1 to 5 (in the first course) were considered to be passing grades and 0 the failing grade. In the second course, the grading was already a pass/fail, so no changes were done to it. In the second dataset, the students were categorized into above median (or equal) and below median. In the first dataset, the data are biased towards passing students (see Table 2), while the second dataset is balanced.

When possible, the features were normalized using min-max normalization [27]. An overview of the features used is shown in Table 3.

3.4 Classifier Evaluation

The applicability of these features for predicting course outcomes was assessed using a set of classifiers. First, three classifiers (Support Vector Machine, Random Forest, and AdaBoost) are compared to assess the performance of traditional machine-learning methods. Two versions of Support Vector Machine are used, one with a linear kernel and one with a radial bases function kernel. Sup-